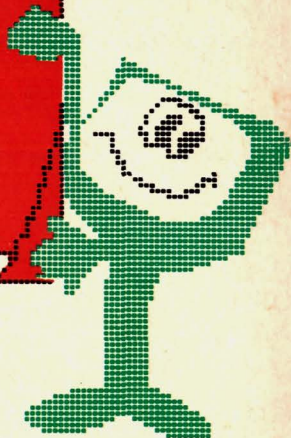
20 VIDEOLEZIONI DI BASIC
PER IMPARARE COL VIC 20

GRUPPO
EDITORIALE
JACKSON

7

COMMODORE
VIC20

VIDEO BASIC VIC 20
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea - Via Indipendenza 88 - Como
Redazione software:
Francesco Franceschini, Enrico Braglia,
Fabio Calanca
Segretaria di Redazione:
Marta Menegardo
Progetto grafico:
Studio Nuovaidea - Via Longhi 16 - Milano
Impaginazione:
Silvana Corbelli
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.r.l. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

Gruppo Editoriale Jackson

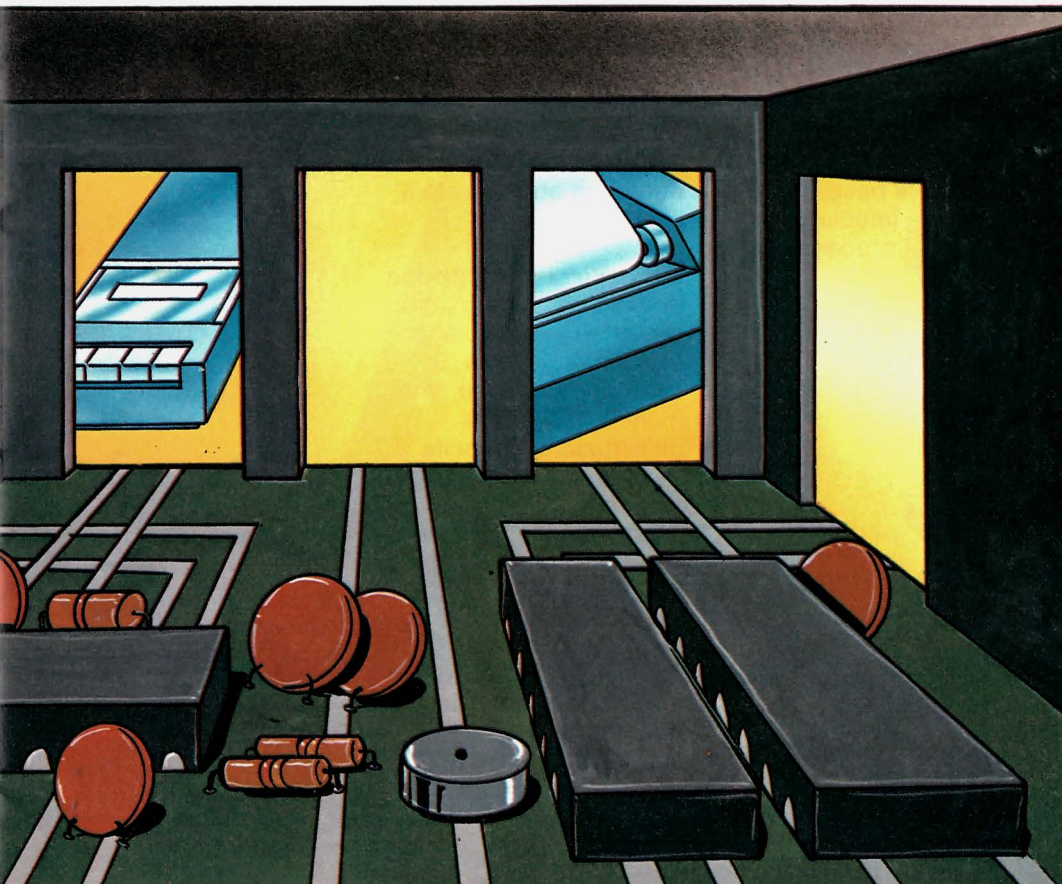
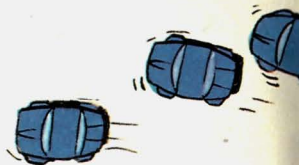
SOMMARIO

Introduzione

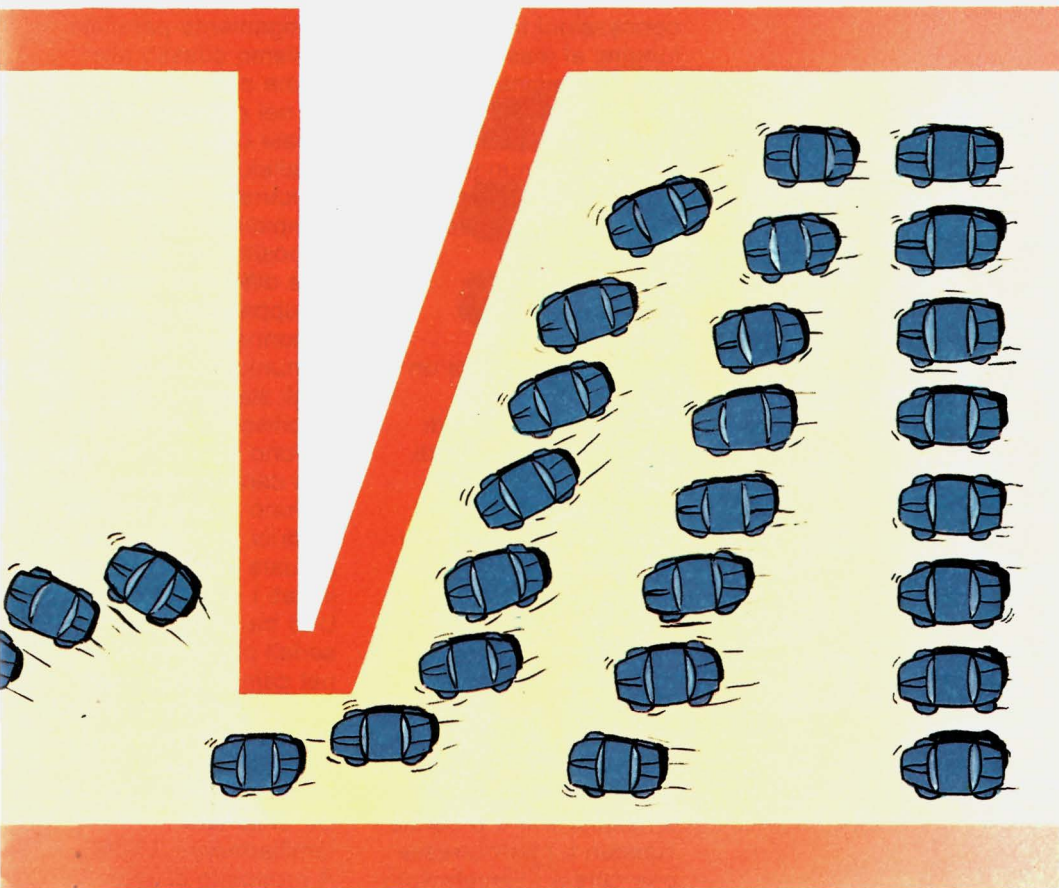
Esistono due tecniche di trasmissione dati: seriale e parallela. La prima è più semplice e economica, la seconda più veloce. Nel tuo VIC 20 coesistono entrambe per permettere la comunicazione tra computer e periferiche e tra parti diverse del computer stesso.
Abbiamo già accennato, poi, a come il computer può perdere delle "quasi decisioni", proseguiamo su questa strada, introducendo gli operatori logici (AND-OR-NOT) che permettono confronti e quindi decisioni, secondo schemi e strutture logiche simili a quelle seguite dall'uomo, nei suoi ragionamenti.
E ancora ... STOP, CONTINUE, NEW, ON ... GOTO, TI, TI$, CLR.

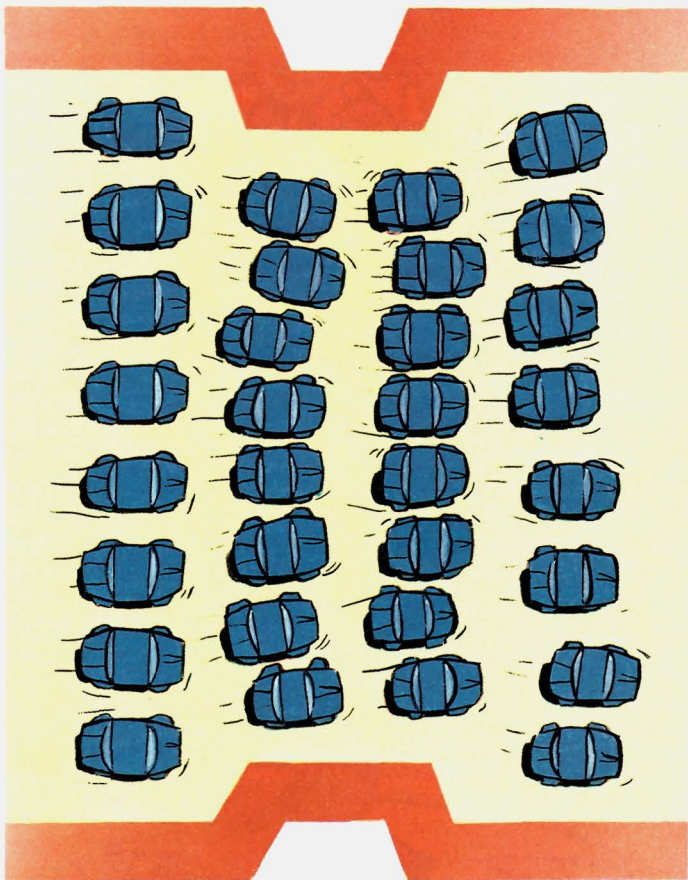
Le porte

Abbiamo visto nelle scorse lezioni (e lo vedremo anche nelle prossime!) come il cuore del calcolatore, cioè la CPU, sia in grado di effettuare - attraverso lunghe, ma velocissime serie di operazioni elementari - tutte le elaborazioni, i calcoli e gli ordinamenti necessari per portare a compimento le diverse sequenze di azioni specificate di volta in volta dal programma presente nella memoria.
Elaborare dati per mezzo di un computer significa appunto raggiungere i diversi risultati attraverso tali serie di operazioni.
Quella che però non è stata forse ancora sufficientemente specificata e sottolineata è l'estrema e fondamentale importanza che in tutto il processo assumono le varie unità periferiche.
Ogni singolo calcolo comporta infatti un insieme di operazioni (di lettura, localizzazione, elaborazione, trasferimento ed emissione), talmente complesso e molteplice che nessuna unità centrale "nuda e cruda", priva cioè di adeguati ed opportuni supporti esecutivi, può essere in grado di affrontare e risolvere.
Proponendoci di esaminare nelle prossime lezioni le descrizioni dettagliate e specifiche delle varie unità periferiche, vogliamo oggi occuparci di un argomento spesso sottovalutato o addirittura ignorato (ma non per questo di secondaria importanza, anzi!): le tecniche di collegamento e comunicazione tra unità centrale e periferica o - detto in termini più generali - di interfacciamento tra unità centrale e mondo esterno.
È chiaro (o almeno dovrebbe esserlo) che qualsiasi "colloquio" tra computer e periferica debba avvenire - indipendentemente dalla periferica - grazie ad un collegamento elettrico.
Dobbiamo quindi vedere per prima cosa il modo in cui tali connessioni possono essere fisicamente realizzate.
Escludendo l'ovvia presenza di uno o più cavi conduttori, col

compito di congiungere l'uno con l'altro i vari elementi, la parte principale del sistema di comunicazione è fondamentalmente costituita dalle cosiddette "porte" di ingresso/uscita.
Una porta non è altro che un circuito elettrico (di solito inserito all'interno del computer) in grado, a seguito di opportune e prefissate condizioni, di far entrare od uscire dall'elaboratore le informazioni che si possono rendere man mano necessarie o disponibili.
Un esempio ti chiarirà immediatamente le idee. Quando batti un qualsiasi tasto sul tuo computer la pressione che hai esercitato sulla tastiera mette immediatamente in azione un particolare dispositivo che, come ben sai, genera la specifica combinazione di bit corrispondente al tasto premuto.
Fino a questo punto la sequenza di bit è però, in un certo senso, esterna all'elaboratore: occorre infatti un ulteriore dispositivo che

"interfacci" (cioè metta appropriatamente in comunicazione) il generatore dei codici dei caratteri con la memoria e l'unità centrale. Tale dispositivo è appunto una porta, in questo caso particolare una porta di ingresso, visto che l'informazione è in entrata: grazie ad essa il carattere premuto riesce così ad arrivare "dentro" il calcolatore, finalmente pronto e disponibile per le eventuali, ulteriori elaborazioni.
Tutto sembrerebbe semplice allora: quando la CPU desidera compiere una operazione di ingresso o di uscita le basta leggere o scrivere (mediante opportune procedure) il dato da scambiare per mezzo della porta corrispondente alla periferica selezionata, e il problema è risolto.
Purtroppo le cose non sono così semplici. La velocità di lavoro dell'unità centrale è infatti ben diversa da quella di una qualsiasi periferica. Se i dati da scambiare sono più di uno (e ciò accade nella stragrande maggioranza dei casi), può capitare che la CPU esegua l'insieme delle operazioni ad una velocità talmente elevata da tentare di leggere o scrivere nella porta ancora prima che la periferica abbia avuto il tempo di "digerire" il dato precedente.
Nel caso di un'istruzione di ingresso questo fatto provoca una doppia (o tripla, o quadrupla ...) lettura dello stesso dato, mentre nel caso di un'istruzione di uscita cambia le carte in tavola alla periferica nel bel mezzo di un'operazione, con risultati chiaramente imprevedibili.
Si rende pertanto necessario un "serbatoio" intermedio, cioè un elemento nel quale tutte le informazioni possano accumularsi senza alcuna conseguenza per il regolare svolgimento delle operazioni.
Questo serbatoio, tecnicamente chiamato buffer o memoria tampone, ha il compito di "assorbire" ad alta velocità i vari dati, "rilasciandoli" uno dopo l'altro, via via che la

periferica è in grado di smaltirli.
Entreremo comunque in ulteriori dettagli nelle prossime lezioni: per il momento è infatti necessario e sufficiente che tu abbia le idee chiare solo sui principi che stanno alla base dei vari processi.

Trasmissione seriale e parallela

Parliamo adesso della realizzazione della connessione tra computer e periferica.
È possibile riunire le tecniche di collegamento in due categorie generali: seriale e parallela.
Nel collegamento seriale i bit che compongono ciascun dato (te ne ricordi? 8 bit = 1 byte) vengono inviati, uno dopo l'altro e sotto forma di impulsi elettrici, lungo uno stesso filo ad

intervalli regolari di tempo.
Attraverso il filo scorrono quindi, opportunamente distanziati tra di loro, "treni" di impulsi, a ciascuno dei quali è fatto precedere un segnale di "arrivo di un dato" che per il circuito adibito alla ricezione ha appunto il preciso significato di prepararsi per l'arrivo del dato.
È un tipo di collegamento molto economico e semplice da realizzare, e pertanto usato assai di frequente nei personal computer.
Non consente però il raggiungimento di elevate velocità di comunicazione, né la realizzazione di connessioni troppo lunghe: al massimo qualche decina di metri.
Questi difetti - se di difetti si può parlare - risultano comunque, nella maggior parte dei casi, assolutamente irrilevanti ai fini dell'utilizzo di un personal computer.
Nel collegamento parallelo c'è invece un filo per ogni bit da trasmettere; il cavo di collegamento è quindi costituito da 8 fili più altri fili ausiliari per utilizzi e comunicazioni di servizio, sui quali non è però assolutamente il caso di soffermarsi.
I bit che compongono i dati da spedire non vengono più inviati l'uno dopo l'altro, ma l'uno accanto all'altro. È un po' come se, in un'autostrada a 8 corsie, ciascun bit percorresse una corsia riservata solo ed esclusivamente ad esso.
Il principale difetto di questa soluzione balza subito agli occhi: la realizzazione hardware non può infatti che influenzare notevolmente il costo di tutta la connessione.
Ciò nonostante molte periferiche utilizzano come sistema di comunicazione il collegamento parallelo.
Veniamo adesso alle norme costruttive.
Sia nel collegamento seriale che in quello parallelo, allo scopo di assicurare un minimo di compatibilità tra dispositivi prodotti da ditte differenti, sono stati sviluppati dei protocolli di connessione e comunicazione, cioè degli standard, che stabiliscono come i fili devono essere collegati, che caratteristiche devono avere i segnali elettrici, in che ordine ed a quale velocità vanno inviati i segnali, ecc..
Uno tra gli standard seriali più utilizzati (anzi, nel campo dei piccoli elaboratori praticamente l'unico) è la versione semplificata del protocollo RS 232 C/V 24, chiamato familiarmente RS 232.
A proposito di velocità di

comunicazione: essa specifica il numero di bit che possono essere inviati ogni secondo lungo la linea.
L'unità di misura è pertanto il bit per secondo (bps), chiamato anche baud rate. Valori tipici della velocità di trasmissione dei dati sono quelli compresi tra i 40 e 19200.
Come hai potuto vedere, l'interfacciamento tra unità centrale ed una periferica è un argomento tutt'altro che di semplice e sbrigativa soluzione. Sono infatti necessari approfonditi studi sulle compatibilità tra i vari elementi, sugli standard ed i protocòlli di comunicazione e sulle procedure costruttive.
Nelle prossime lezioni, quando cioè parleremo in dettaglio delle singole periferiche, ritorneremo comunque più specificatamente sull'argomento.

Gli operatori logici

Abbiamo più volte detto, come la grande diffusione che i computer hanno avuto, e stanno attualmente avendo, dipenda in gran parte dalla loro estrema velocità e precisione nell'effettuare calcoli e confronti.
Soprattutto i confronti consentono spesso e volentieri agli elaboratori di avvicinare ed emulare un modo di agire ed operare che all'uomo risulta particolarmente semplice, consueto e familiare: il confronto, cioè un ragionamento secondo schemi e strutture logiche.
In questo ambito, quindi, gli uomini e le macchine si trovano perfettamente d'accordo: ciò che di solito è più semplice per l'uomo lo è infatti anche per la macchina, e viceversa.
Tutti i linguaggi di programmazione annoverano e contemplano pertanto come una delle istruzioni più potenti ed importanti quella che consente materialmente di eseguire i confronti e le decisioni. Grazie ad essa è infatti possibile programmare seguendo lo stesso modello logico - normalmente basato su una serie di mutue esclusioni - che in fondo qualsiasi essere umano è sempre solito seguire ogniqualvolta si trova nelle condizioni di dover decidere tra un certo insieme di scelte o

possibilità.
Per esempio, la decisione conseguente ad un ragionamento del tipo: "Domani andrò a sciare, se ci sarà bel tempo, se Mario verrà insieme a me e se riuscirò ad alzarmi presto. Altrimenti andrò a farmi una bella passeggiata", non potrà che essere raggiunta ed attuata soltanto dopo che saranno stati analizzati e valutati, uno per uno, tutti i fattori sui quali il ragionamento si basa e dai quali dipende interamente.
È quindi necessario strutturare in maniera analoga anche le istruzioni dei programmi corrispondenti a tali decisioni.

Sinora, però, tutte le espressioni condizionali (o relazionali) che abbiamo incontrato erano di tipo semplice: potevamo cioè operare sugli elementi della condizione (o della relazione) soltanto con una distinzione del tipo vero-falso, si-no, maggiore-minore, uno-zero.
Si rende pertanto necessario, per essere in grado di eseguire all'interno delle espressioni condizionali e logiche tutte le operazioni che possono diventare utili o necessarie, poter disporre di una nuova "famiglia" di operatori: i cosiddetti operatori logici.
Finora avevamo infatti conosciuto ed utilizzato soltanto gli operatori aritmetici (con i quali elaborare le espressioni matematiche) e gli operatori relazionali (mediante cui eseguire i confronti). Gli operatori logici vengono invece usati per combinare in un unico risultato logico due distinti valori logici.
Gli operatori logici sono:

AND (l'uno e l'altro)
OR (l'uno o l'altro)
NOT (negazione)

Un confronto del tipo: 5 > 3 è verificato, cioè è vero; all'espressione 5 > 3 il tuo computer assegna convenzionalmente il valore − 1.
La condizione 8 < 5 è invece non verificata, cioè è falsa; all'espressione 8 < 5 compete quindi (sempre per convenzione) il valore 0.
Sfruttando questo fatto è pertanto possibile scrivere istruzioni del tipo:

```
LET A = (5 > B)
```

Alla variabile A viene assegnato il valore 0 o — 1, in dipendenza del fatto che B risulti maggiore o minore di 5.
Oppure:

```
LET K = (4 = A)
```

K varrà 0 o − 1, se A risulterà diverso od uguale a 4.

NOT

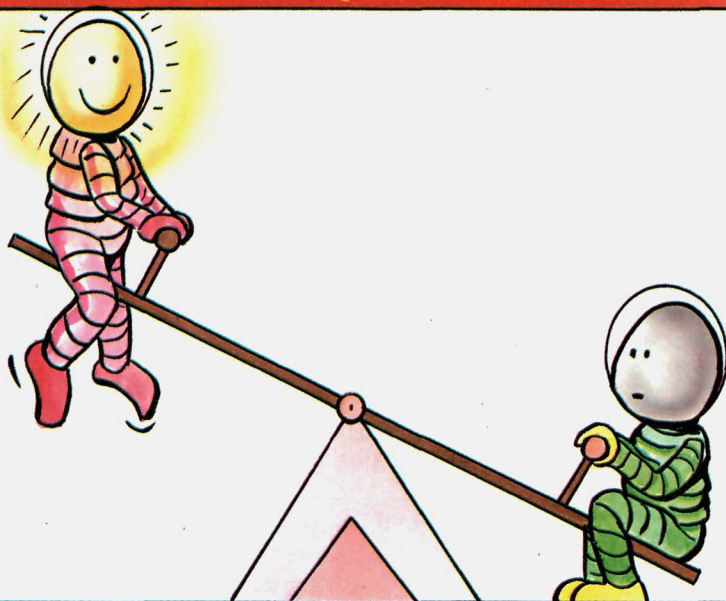
NOT è la più elementare operazione logica; essa significa negazione, cioè inversione dello stato di una variabile o di una relazione.

Così, se A è una variabile di valore 0, la sua negazione (NOT A) vale — 1. Se invece A assume un qualsiasi altro valore il tuo computer eseguirà un'operazione chiamata complemento A2 del numero binario.
In pratica il valore di NOT A sarà determinato dall'espressione
— (A + 1).

Ad esempio:

NOT − 7 = − (− 7 + 1) = − (− 6) = 6

Esempi

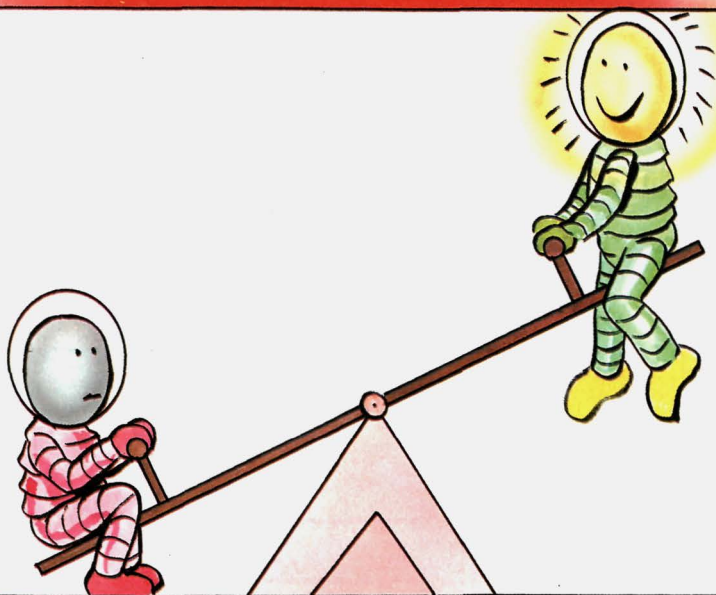
LET S = NOT 5	Assegna a S il valore — 6
LET J = NOT 0	Assegna a J il valore − 1
LET A = (NOT (7 > 3)) < > 0	7 è maggiore di 3, quindi 7 > 3 è considerato − 1. NOT − 1 è uguale a 0. Poiché zero non è diverso da zero, assumerà quindi valore 0.

La tabella della verità

Spesso torna utile far ricorso alla cosiddetta tabella della verità. Questa non è altro che una schematica rappresentazione dell'operatore logico, avente come scopo quello di fornire tutti i risultati corrispondenti alle diverse possibili combinazioni dell'espressione, precisando e visualizzando rapidamente le relazioni intercorrenti tra valori in ingresso e risultati in uscita.
La tabella della verità di NOT è molto semplice:

ingresso	uscita
1	0
0	1

OR

L'OR, o somma logica, viene invece definito in questo modo: "la somma logica tra due espressioni logiche è vera (cioè uguale ad 1), se anche una sola di esse è vera (cioè 1); se entrambe sono false (uguali a 0), la somma logica è falsa".
Se allora definiamo la variabile C in questo modo:

```
LET C = (A > 1) OR (B > 2)
```

ad essa il tuo computer assegnerà valore − 1, se almeno una delle due espressioni entro parentesi risulterà verificata, cioè uguale a 1. Se invece nessuna delle due parentesi sarà vera, C varrà 0.

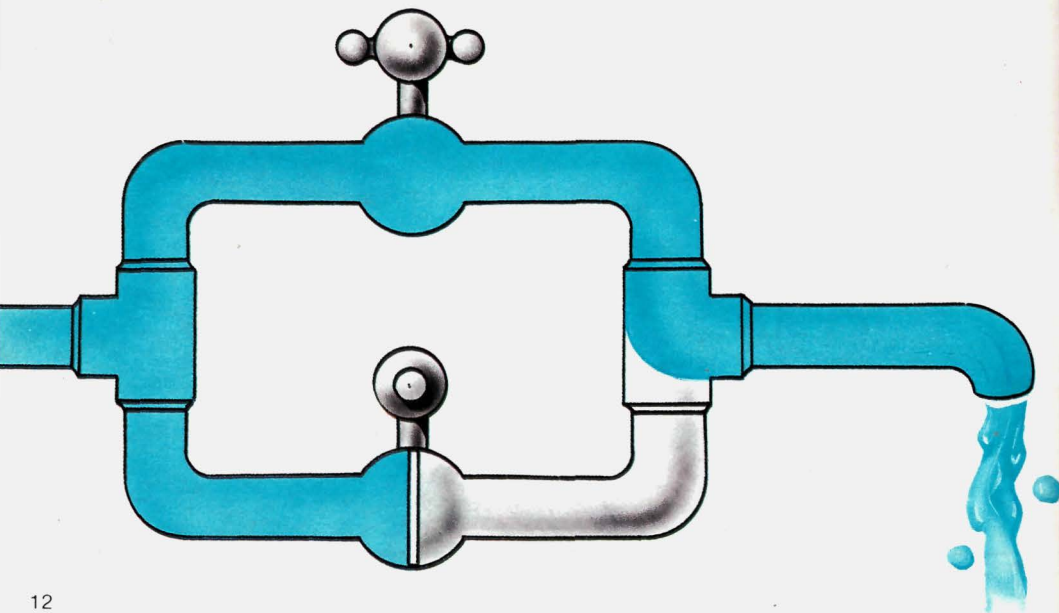
```
LET J = (5 > D) OR (Z = 9)
```

J varrà − 1, se D risulterà minore di 5 o se Z sarà pari a 9. Varrà 0 in caso contrario.

La tabella della verità

La tabella della verità di OR è la seguente (A e B sono due qualsiasi espressioni logiche):

A	B	A OR B
0	0	0
1	0	1
0	1	1
1	1	1

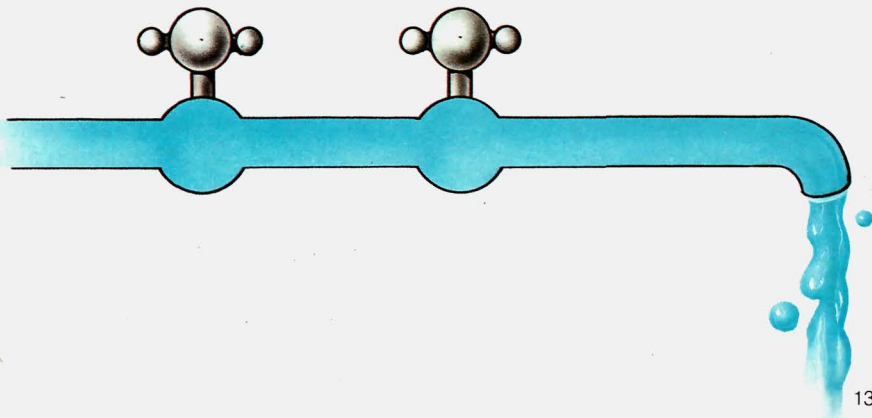
Come puoi vedere, l'unico caso che può rendere A OR B pari a zero è quello in cui entrambe le espressioni risultano false, cioè pari a zero.

AND

L'ultimo operatore logico che ci resta da esaminare è AND (o prodotto logico). Esso è definito nel seguente modo: "il prodotto logico tra due espressioni logiche è vero (quindi pari ad 1), se e solo se entrambe le espressioni sono vere (uguali ad 1)".
La variabile F, definita come

```
LET F = (A > 1) AND (B > 2)
```

risulterà pertanto pari ad 1 soltanto se entrambe le espressioni all'interno delle parentesi risulteranno verificate, cioè se A sarà maggiore di 1 e B maggiore di 2.
Come al solito il tuo computer assegnerà alle espressioni verificate il valore − 1.
Se invece anche una sola delle due parentesi conterrà un'espressione falsa, F assumerà valore zero.

```
LET I = (G = 3.1) AND (T = 4.5)
```

I assumerà valore − 1 se G e T avranno valore rispettivamente uguale a 3.1 e 4.5. In caso contrario il valore sarà 0.

La tabella della verità

Questa volta l'unico modo per ottenere un risultato vero è che entrambe le espressioni risultino vere, cioè 1.
In un certo senso si può quindi dire che il prodotto logico è l'operazione inversa alla somma logica (se provi a confrontare le due tabelle AND OR, te ne renderai subito conto).

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Se OR e AND sono usati invece che per confronti, come operatori logici tra valori numerici, il risultato ottenuto è dato dall'operazione logica applicata ai singoli bit del corrispondente numero binario.
Ad esempio: 127 OR 3 ti restituirà il valore 127, perché il corrispondente binario del decimale 127 è 01111111, quello di 3 è 00000011 e la OR logica di ogni singolo bit ricavata dalla tabella della verità dà come risultato 01111111 che equivale al decimale 127.

0	1	1	1	1	1	1	1
OR							
0	0	0	0	0	0	1	1
RISULTATO							
0	1	1	1	1	1	1	1

4 AND 1 ti restituirà il valore 0.

0	0	0	0	0	1	0	0
AND							
0	0	0	0	0	0	0	1
RISULTATO							
0	0	0	0	0	0	0	0

Priorità degli operatori logici

Anche gli operatori logici hanno una loro priorità, analogamente agli operatori matematici.
L'ordine delle priorità nelle operazioni logiche è infatti, partendo dalla più bassa alla più alta, OR, AND, NOT.
Tali precedenze possono naturalmente sempre essere alterate facendo uso delle parentesi: anzi, visto che l'occhio umano non è molto abituato a leggere espressioni come

```
A = NOT C > A AND B OR D AND C
```

sarà meglio, per evitare facili errori o confusioni, utilizzare sempre e comunque le parentesi:

```
A = (NOT (C > A) AND B) OR (D AND C)
```

Nella parte della lezione dedicata alla programmazione troverai comunque alcuni esempi applicativi, che ti consentiranno di acquisire maggiore pratica e familiarità sull'uso degli operatori logici.

STOP

Immagina di essere in una sala di regia seduto davanti alla moviola: stai verificando la sequenza delle immagini di un film. All'improvviso ti accorgi che manca la sequenza centrale (quella che spiega "i risvolti psicologici" del protagonista).
Interrompi allora la proiezione, cerchi e (per fortuna) trovi il pezzo mancante, senza il quale il film sarebbe risultato un "polpettone", e lo incolli nel punto esatto in cui era previsto.
Ora è tutto a posto e la proiezione può proseguire.

Trasferisci adesso tutto il discorso su un altro piano: il film è un programma e tu, da operatore, diventi un programmatore.
Il programma non funziona nel migliore dei modi: per quanto esso sembri non evidenziare alcun errore lampante e palese, ad un certo punto si blocca con un messaggio di errore.
Ti farebbe proprio comodo un'istruzione che potesse arrestare momentaneamente

l'esecuzione, consentendoti di controllare alcuni risultati intermedi e di riprendere come se niente fosse accaduto ... Che fare?
Il BASIC, per i casi come questo, ti mette a disposizione l'istruzione STOP.
Infatti STOP, ferma l'esecuzione di un programma e provoca la visualizzazione di un messaggio indicante la linea di programma nella quale l'interruzione è avvenuta, proprio come se si fosse verificato un errore.
Il grosso vantaggio è che dopo una STOP il calcolatore si pone nel modo "editor": puoi cioè inserire nuove linee, apportate correzioni, esaminare o modificare il valore delle variabili.

Naturalmente, una volta risolti tutti i problemi è bene che gli STOP siano eliminati. A nessuno, fa piacere vedere un programma arrestarsi nel bel mezzo dell'esecuzione!
Così, per esempio, la linea

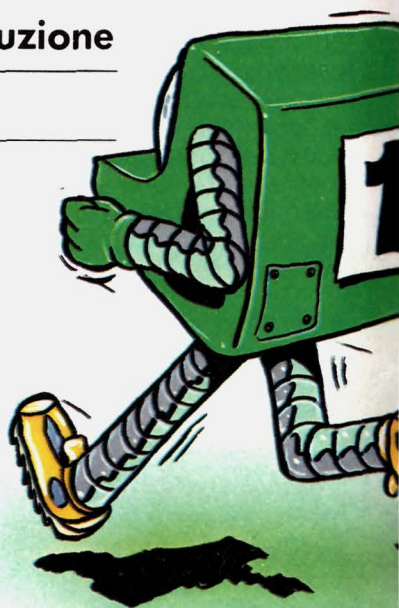
```
150 STOP
```

visualizzerà, quando eseguita:

```
BREAK IN 150
```

avvertendoti che l'interruzione del programma si è verificata alla linea 150, in seguito ad un'istruzione STOP.

Sintassi dell'istruzione

STOP

CONTINUE

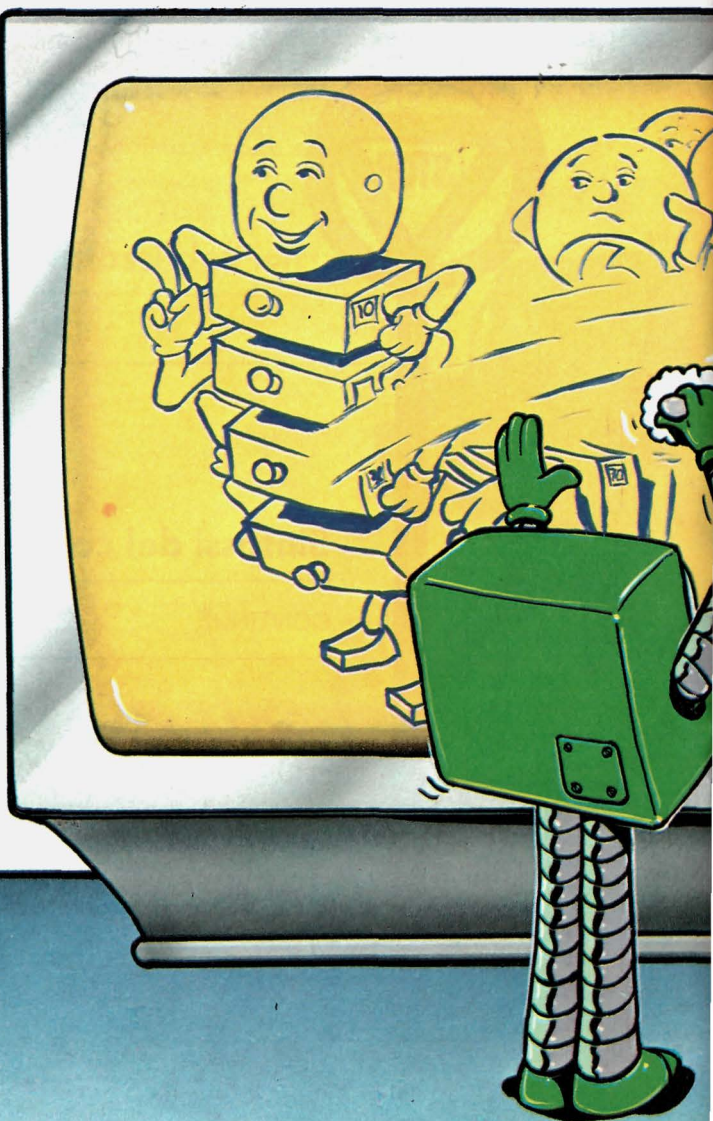
Come alla moviola è possibile riprendere la proiezione dal punto esatto di interruzione, anche in un programma che è stato bloccato in seguito ad una STOP è possibile riprendere l'esecuzione, grazie al comando CONTINUE (che ovviamente va impartito in modo immediato).
STOP e CONTINUE costituiscono pertanto un utilissimo strumento per il debugging, cioè per la correzione e la messa a punto dei programmi.
Con essi puoi infatti controllare i punti strategici di un programma, eventualmente modificando e migliorando quelle istruzioni che in un primo tempo ti sembravano perfettamente corrette, senza però per questo interrompere in modo definitivo l'elaborazione che era in corso fino a quel momento.

Sintassi del comando

CONTINUE

NEW

Talvolta, hai la necessità di cancellare un programma dalla memoria del computer. In questo caso due sono le possibili alternative: togliere semplicemente l'alimentazione al computer, per poi riaccenderlo, oppure - con molta più eleganza - impartire il comando NEW.

NEW serve infatti per cancellare il programma BASIC presente in memoria insieme a tutte le variabili. Normalmente lo si usa quando si inizia a scrivere un nuovo programma e si vuole essere sicuri che non vi sia alcuna istruzione, o

parte di programma, già presente in memoria.
È naturalmente un comando da utilizzare con molta attenzione: un NEW impartito senza troppa cura può infatti provocare in un attimo la scomparsa di ore ed ore di lavoro alla tastiera.

NEW viene solitamente adoperato in esecuzione immediata; tuttavia lo puoi inserire anche all'interno di un programma:

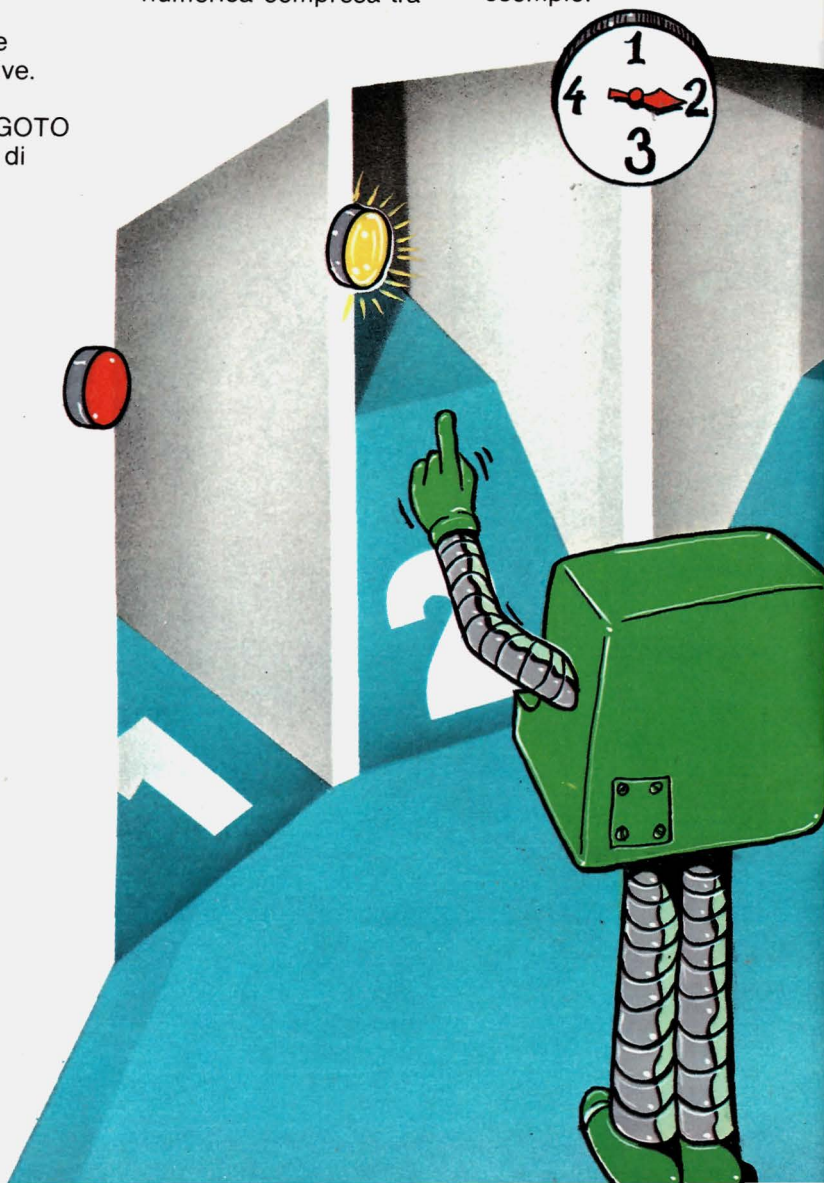
```
190 REM ATTENZIONE: RISPONDENDO
    CON NO PERDI IL PROGRAMMA
200 IF A$ = "NO" THEN NEW
```

In questo caso, però, devi tener ben presente che, al momento dell'esecuzione della linea 200, l'intero programma verrà cancellato, provocando come conseguenza l'istantaneo arresto di *qualsiasi attività* del calcolatore.
Sarà comunque buona cosa che prima di inserire questo comando in un programma tu impari a utilizzarlo ed a capirne l'esatto funzionamento nel più classico (e sicuro) utilizzo immediato.

Sintassi del comando

NEW

ON ... GOTO

Abbiamo visto nelle scorse lezioni come l'istruzione IF ti permetta di prendere una decisione tra due possibili alternative. L'istruzione ON <espressione> GOTO consente invece di passare il controllo dell'esecuzione ad una qualsiasi linea di programma, in funzione del valore assunto dall'espressione numerica compresa tra le parole ON e GOTO. ON GOTO consente, pertanto, la realizzazione di una struttura alternativa multipla; vediamo insieme un esempio:

```
10 INPUT A
20 ON A GOTO 40, 50, 60, 70
30 GOTO 10
40 PRINT "HAI PREMUTO 1" : END
50 PRINT "HAI PREMUTO 2" : END
60 PRINT "HAI PREMUTO 3" : END
70 PRINT "HAI PREMUTO 4" : END
```

Il programma inizia richiedendo in ingresso un valore numerico. Valuta quindi l'espressione compresa tra ON e GOTO (nel nostro caso semplicemente costituita da A) e - a seconda che la parte intera di A abbia valore 1, 2, 3, 4 - esegue un salto al primo, al secondo, al terzo od al quarto numero di linea presente dopo la parola GOTO.
Se il valore di A è invece zero, oppure supera la quantità di numeri di linea presenti nell'istruzione, il programma prosegue con la linea di programma immediatamente seguente. Proprio questa è la ragione per cui è stata inserita la riga 30: essa provocherà infatti la riesecuzione del programma fintanto che dalla tastiera sarà stato battuto un valore compreso tra 1 e 4.
Un valore negativo dell'espressione provocherà invece il messaggio:

```
? ILLEGAL QUANTITY
```

L'istruzione ON ... GOTO può essere utile solo quando le differenti possibilità alle quali sei interessato hanno valori consecutivi, per di più inizianti con il valore uno, oppure siano riducibili a tale situazione. È un

comando potente che richiede però una certa pratica di programmazione.
Pensa, infatti, a quanta efficienza ed eleganza e leggibilità ha questa istruzione:

```
100 ON FLAG GOTO 1000, 2000, 3000, 4000, 5000
```

al posto di

```
100 IF FLAG = 1 THEN GOTO 1000
110 IF FLAG = 2 THEN GOTO 2000
120 IF FLAG = 3 THEN GOTO 3000
130 IF FLAG = 4 THEN GOTO 4000
140 IF FLAG = 5 THEN GOTO 5000
```

Senza pensare poi alla maggiore velocità di esecuzione!!

Sintassi dell'istruzione

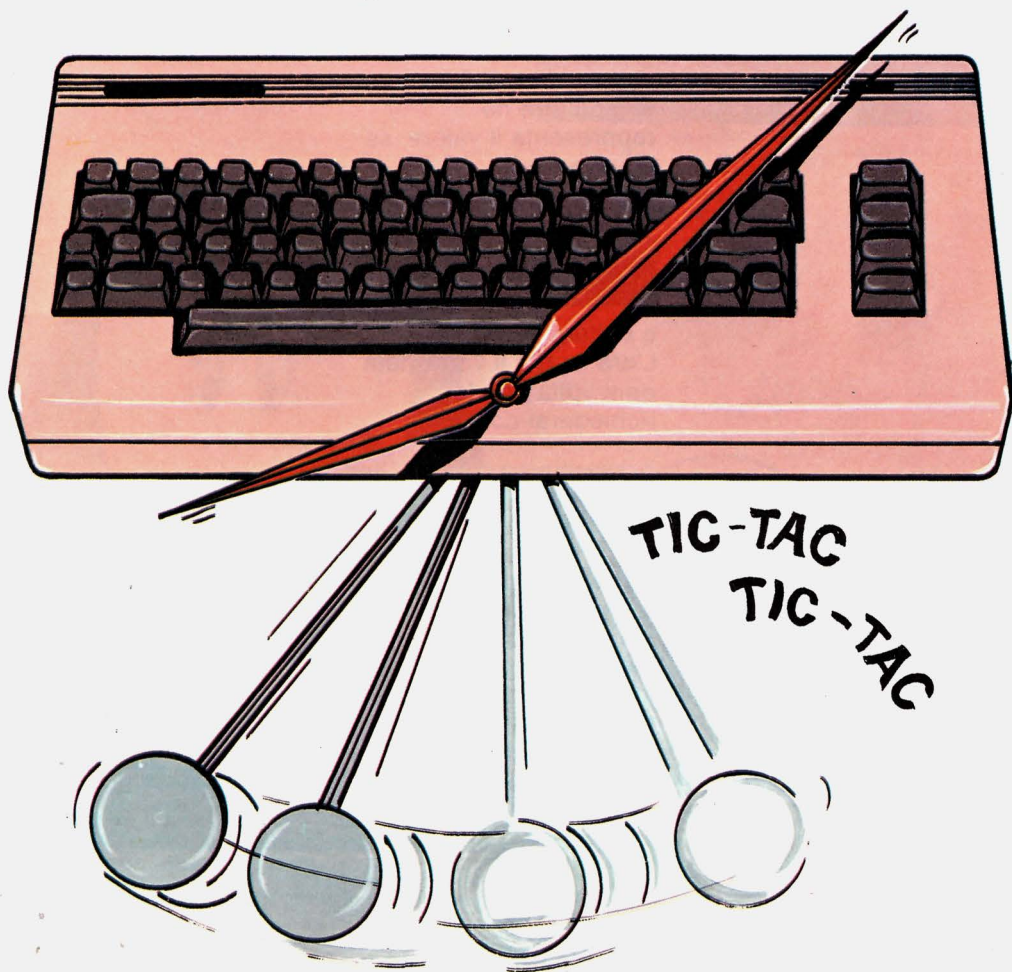
ON espressione GOTO linea1 [,linea2] [, ...]

TI

Il tuo VIC 20 ha uno speciale orologio che parte al momento dell'accensione e misura in sessantesimi di secondo il tempo

trascorso da quell'istante.
Se desideri dunque sapere da quanto tempo stai operando al computer, non devi far altro che introdurre:

```
PRINT TI
```

Dividi il numero ottenuto per 60 ed otterrai il tempo in secondi o per 3600 per averlo in minuti e così via.
Questa funzione può essere molto utile quando ti occorre una misura del tempo molto precisa.

TI$

Con la funzione TI$ puoi invece visualizzare un vero e proprio orologio rappresentato da una stringa di 6 caratteri nella forma "OOMMSS". Ad esempio, un valore di TI$ pari a 103245 significa ore 10 : 32 minuti : 45 secondi, mentre 170000 sta per le 5 pomeridiane.
Per introdurre l'ora esatta, è sufficiente assegnare a TI$ la stringa che ne rappresenta il valore; se adesso fossero le 21 : 15 : 00, dovresti digitare:

```
LET TI$ = "211500"
```

e battere RETURN.
L'ora esatta ti verrà data ogni volta che lo richiederai con:

```
PRINT TI$.
```

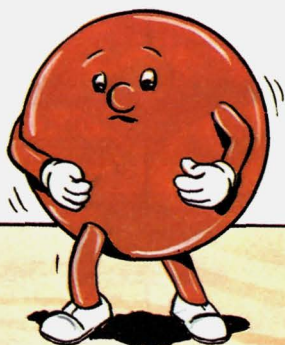
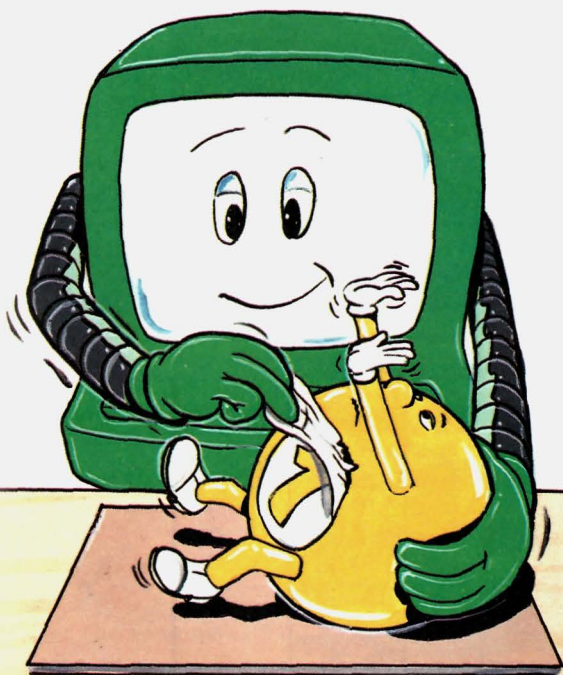
Attenzione però, perché le operazioni di caricamento e di salvataggio da cassetta interferiscono col funzionamento dell'orologio del tuo VIC 20 sospendendone quasi il funzionamento.

CLR

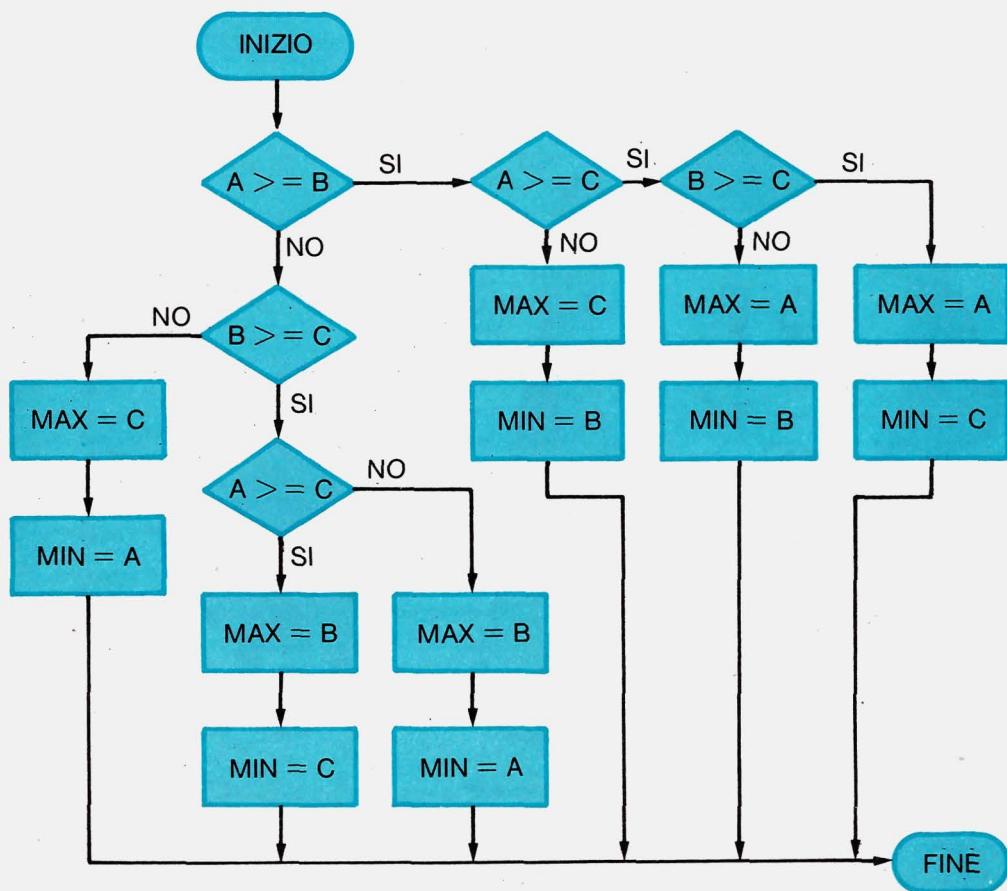
Il comando CLR può essere utilizzato sia in modo immediato che come istruzione in un programma.
Ha il compito di cancellare il valore di tutte le variabili, di qualunque tipo esse siano.

Dopo un CLR, tutte le variabili numeriche avranno valore 0 e quelle stringa valore nullo.

Lascia intatto, però, il programma in memoria ed in questo si differenzia dal più drastico NEW.
Ecco un esempio per testare l'effetto di CLR:

```
10 LET A = RND (0)
20 LET B$ = "PROVA CLR"
30 PRINT A, B$
40 CLR
50 PRINT A, B$
60 LIST
```

Applicazione degli operatori logici

Ci proponiamo di risolvere il seguente problema: dati in ingresso tre numeri qualsiasi, trovarne il maggiore ed il minore. Supponendo di chiamare A, B, C le variabili adibite all'inserimento nel calcolatore dei tre valori numerici, un possibile schema a blocchi potrebbe essere:

L'estrema semplicità del problema non richiede di certo alcuna spiegazione sul flowchart.
Qualche parola è invece necessario spenderla sulla conversione del diagramma nel corrispondente programma BASIC di qui troverai più avanti due distinte versioni.
Il tuo computer le

gradirà entrambe: dal suo punto di vista, infatti, una volta che la sintassi risulta corretta non esiste alcuna differenza tra l'una o l'altra.
Noi invece, ci accorgiamo immediatamente della differente impostazione, struttura e leggibilità intercorrente tra i due listati. Grazie all'uso intensivo degli operatori logici, seguire, comprendere e - soprattutto - controllare la funzione, l'esecuzione e lo svolgimento di ogni singola istruzione della seconda versione dovrebbe risultare difatti molto più semplice e lineare di quanto non sia possibile fare con la prima.
Ciò è dovuto in gran parte all'assenza di istruzioni GOTO, il cui difetto è spesso quello di interrompere la trama logica che si era seguita al momento della stesura dello schema a blocchi.
Un consiglio quindi: impara da subito a programmare, cercando di seguire questa impostazione e diventerai un ottimo programmatore.

```
10 INPUT A, B, C
20 IF A >= B AND A >= C AND B >= C THEN MAX = A : MIN = C
30 IF A >= B AND A >= C AND NOT (B >= C) THEN MAX = A : MIN = B
40 IF A >= B AND NOT (A >= C) THEN MAX = C : MIN = B
50 IF NOT (A >= B) AND NOT (B >= C) THEN MAX = C : MIN = A
60 IF NOT (A >= B) AND B >= C AND A >= C THEN MAX = B : MIN = C
70 IF NOT (A >= B) AND B >= C AND NOT (A >= C) THEN MAX = B :
   MIN = A
80 IF MAX = MIN THEN PRINT "VALORI UGUALI" : END
90 PRINT "IL VALORE MASSIMO È"; MAX
100 PRINT "IL VALORE MINIMO È"; MIN
110 END
```

```
10 INPUT A, B, C
20 IF A >= B THEN GOTO 40
30 GOTO 80
40 IF A >= C THEN GOTO 60
50 MAX = C : MIN = B : GOTO 120
60 IF B >= C THEN MIN = C : MAX = A : GOTO 120
70 MIN = B : MAX = A : GOTO 120
80 IF B >= C THEN GOTO 100
90 MAX = C : MIN = A : GOTO 120
100 IF A >= C THEN MAX = B : MIN = C : GOTO 120
110 MAX = B : MIN = A
120 IF MAX = MIN THEN PRINT "VALORI UGUALI" : END
130 PRINT "IL VALORE MASSIMO È"; MAX
140 PRINT "IL VALORE MINIMO È"; MIN
150 END
```

A proposito: un'ultima osservazione, questa volta riguardo alla sintassi. Cosa succederebbe se per esempio alla riga

```
30 IF A >= B AND A >= C AND NOT (B >= C) THEN MAX = A : MIN = B
```

non venissero messe le parentesi?

```
30 IF A >= B AND A >= C AND NOT B >= C THEN MAX = A : MIN = B
```

Prova a pensarci!

Applicazione dell'istruzione ON ... GOTO

Il problema che ci proponiamo di risolvere è il seguente: assegnata in ingresso una data nel formato GG, MM, AA (dove GG, MM e AA sono valori numerici, come per esempio 13, 03, 85), desideriamo che in uscita la parte MM della data venga trasformata da numerica in alfanumerica, sostituendo cioè il valore indicante il mese con il corrispondente nome italiano del mese (nell'esempio dovremmo quindi ottenere 13 marzo 85).
Ciò viene fatto utilizzando proprio l'istruzione ON ... GOTO:

```
 10 REM TRASFORMA IL MESE
 20 INPUT "GIORNO, MESE, ANNO"; G, M, A
 30 IF M < 1 OR M > 12 THEN PRINT "ERRORE NEL MESE" : GOTO 20
 40 IF G < 1 OR G > 31 THEN PRINT "ERRORE NEL GIORNO" : GOTO 20
 50 ON M GOTO 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170
 60 MESE$ = "GENNAIO" : GOTO 180
 70 MESE$ = "FEBBRAIO" : GOTO 180
 80 MESE$ = "MARZO" : GOTO 180
 90 MESE$ = "APRILE" : GOTO 180
100 MESE$ = "MAGGIO" : GOTO 180
110 MESE$ = "GIUGNO" : GOTO 180
120 MESE$ = "LUGLIO" : GOTO 180
130 MESE$ = "AGOSTO" : GOTO 180
140 MESE$ = "SETTEMBRE" : GOTO 180
150 MESE$ = "OTTOBRE" : GOTO 180
160 MESE$ = "NOVEMBRE" : GOTO 180
170 MESE$ = "DICEMBRE"
180 PRINT "LA DATA IMMESSA È : "; G ;" "; MESE$; " "; A
190 END
```

Il programma comincia richiedendo in ingresso i 3 valori del giorno, del mese e dell'anno (riga 20).
Le linee 30 e 40 controllano quindi che al mese ed al giorno siano stati assegnati valori ammissibili. Nota come l'operatore logico OR abbia semplificato la stesura delle istruzioni condizionali; anziché dover scrivere:

```
IF M < 1 THEN PRINT "ERRORE NEL MESE" : GOTO 20
IF M > 12 THEN PRINT "ERRORE NEL MESE" : GOTO 20
IF G < 1 THEN PRINT "ERRORE NEL GIORNO" : GOTO 20
IF G > 31 THEN PRINT "ERRORE NEL GIORNO" : GOTO 20
```

si è infatti potuto ricorrere a sole due righe, permettendo di

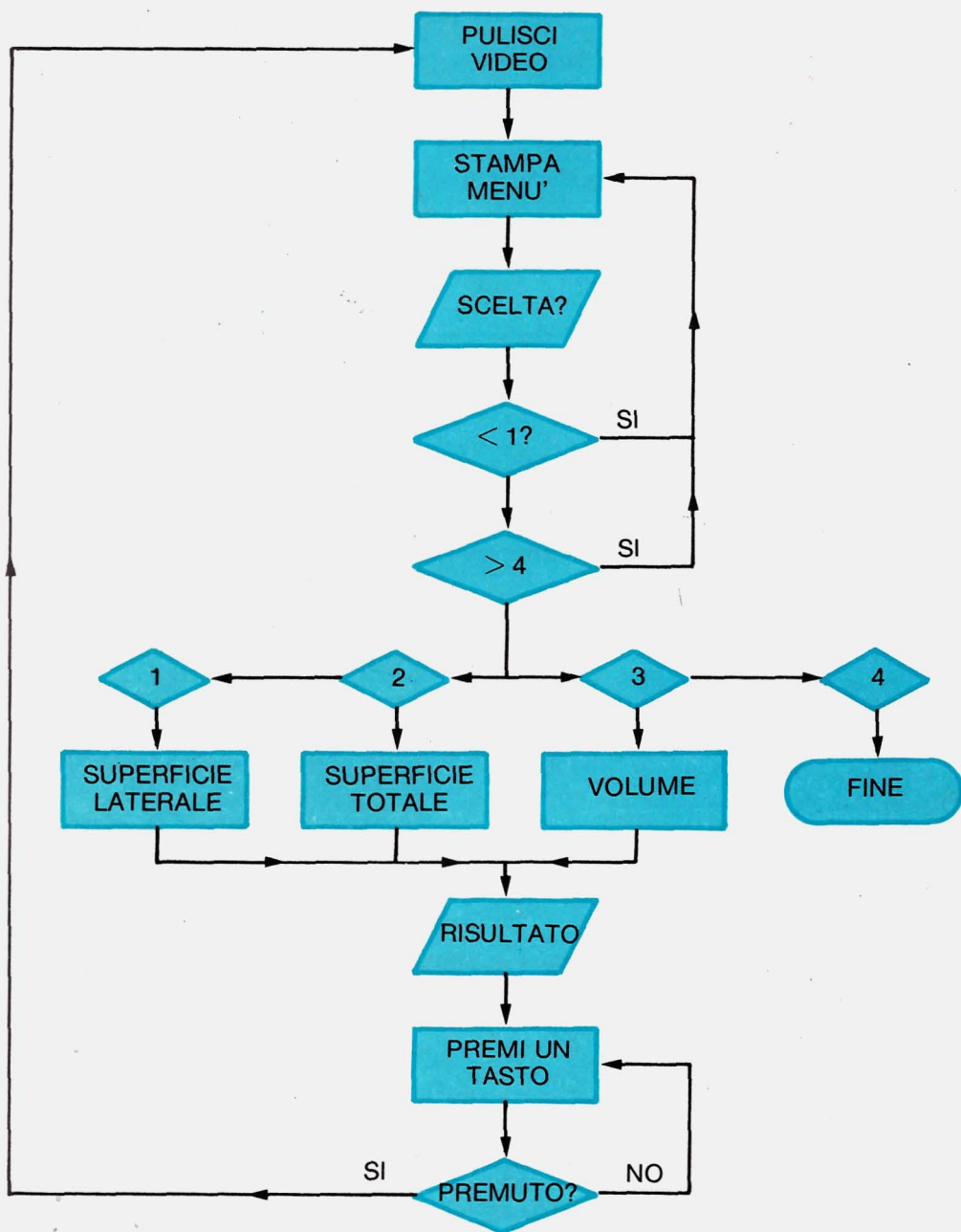
PROGRAMMAZIONE

conseguenza un risparmio di tempo e memoria del computer.
Alla linea 50 appare finalmente la nostra amica ON ... GOTO: a seconda del valore attribuito ad M, essa passerà quindi il controllo ad una delle istruzione di assegnazione specificate dai numeri di linea successivi al ON ... GOTO (abbiamo omesso dalla linea 60 alla 170 l'opzionale LET; comunque è come se ci fosse).
La stampa della data nel formato richiesto (riga 180) concluderà quindi l'esecuzione del programma.

Programma a menù

Quando, come nel programma seguente, si offre all'utente una scelta tra diverse possibilità, è bene ricorrere al menù, cioè ad una lista delle varie opzioni, opportunamente presentate.
In questi casi torna particolarmente utile l'istruzione ON ... GOTO poiché consente di limitare il numero delle IF di verifica della scelta effettuata. Ne guadagna la semplicità e la chiarezza del programma.

```
10 PRINT "[C][Q][]][]][]][]][]][]][]]IL CUBO[Q]"
15 PRINT "1 SUPERFICIE LATERALE"
20 PRINT "2 SUPERFICIE TOTALE"
25 PRINT "3 VOLUME"
30 PRINT "4 FINE"
35 INPUT "[Q] QUALE SCEGLI"; A
40 IFA < 1 ORA > 4 THEN GOTO 10
45 IFA < 4 THEN INPUT "[Q]SPIGOLO"; S
50 ON A GOTO 55, 60, 65, 70
55 LET R = S ↑ 2 * 4 : GOTO 75
60 LET R = S ↑ 2 * 6 : GOTO 75
65 LET R = S ↑ 3 : GOTO 75
70 END
75 PRINT "[Q]RISULTATO ="; R
80 PRINT "[Q]PREMI UN TASTO"
85 GET A$ : IFA$ = " " THEN GOTO 85
90 GOTO 10
```

PULISCI VIDEO
STAMPA MENU'
SCELTA?
< 1?
SI
> 4
SI
1
2
3
4
SUPERFICIE LATERALE
SUPERFICIE TOTALE
VOLUME
FINE
RISULTATO
PREMI UN TASTO
SI
PREMUTO?
NO

VIDEOESERCIZI

Annota nello spazio apposito il risultato da te previsto per ciascun esercizio proposto e poi verificalo con la soluzione del tuo computer. Se avrai commesso anche un solo errore, ripassa la lezione.

```
10 LET A = 12 : LET B = 5 : LET N$ = "BENE"
20 IF A < B THEN PRINT "MELE"
30 IF B < A THEN PRINT "BANANE"
40 IF (A + B) < > (B + A) THEN PRINT N$
```

Quale è la materia considerata più importante in questo listato?

```
10 PRINT "♥CHE VOTO MERITI?"
20 INPUT "TEORIA"; T
30 INPUT "PROGRAMMAZIONE"; P
40 LET M = T + P/2
50 IF P > = 7 AND M > 6 THEN PRINT "PROMOSSO"
60 IF P < 7 OR M < = 6 THEN PRINT "RESPINTO"
```

Secondo te, con 3 in teoria e 8 in programmazione si è promossi?

Quanto volte verrà eseguita l'elaborazione di questo Loop?

```
10 FOR I = 1 TO 10
20 PRINT I : CLR
30 NEXT I
```

Perché?

Per chi ama nascondere i propri segreti ...

```
10 PRINT CHR$ (147) : REM = A PRINT " ♥ "
20 POKE 36879,41
30 PRINT "+ È UN SEGRETO"
40 FOR I = 1 TO 1000 : NEXT ; REM PAUSA
50 PRINT "E Q Q J J NON È UN SEGRETO"
```

Una tecnica usata per nascondere i listati da sguardi indiscreti è quella di stamparli con inchiostro uguale al colore dello sfondo (carta).